2.1进程与线程（上）

进程的概念和特征

进程的概念

- 为了更好地描述和控制程序的并发执行，实现操作系统的并发性和共享性
- 进程控制块（PCB）：为了更好的描述进程的基本情况和运行状态，进而控制和管理进程
  - PCB是进程存在的唯一标志
- 进程的一些典型定义
  - 进程是程序一次执行过程
  - 进程是一次程序及其数据在处理机上顺序执行时所发生的活动
  - 进程是具有独立功能的程序在一个数据集合上运行的过程，是资源分配和调度的独立单位（没有引入线程）

进程的特征

- 动态性：动态性是进程最基本特征，进程有着创建、活动、暂停、终止等过程，具有生命周期
- 并发性：多个进程实体同时存在内存中，引入进程的目的就是为了程序与其他程序并发执行
- 独立性：进程实体是一个能独立运行、独立获得资源和独立接受调度的基本单位
  - 没有建立PCB的程序，都不能作为一个独立单位参与运行
- 异步性：进程相互制约，进程以不可预知的速度向前推进
  - 所以操作系统中一定要配置响应的进程同步机制
- 结构性：每个进程都配置一个PCB对其进行描述
  - 程序段
  - 数据段
  - 进程控制段

进程实体

进程的状态与转换

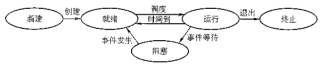
状态

- 运行态：进程在处理机上运行
- 就绪态：进程已处于准备运行状态
- 阻塞态：又称等待态，进程正在等待某个时间而暂停运行
- 创建态：进程正在被创建，尚未进入就绪态
- 结束态：进程正在从系统中消失（包括正常结束或者异常终止）

相互转换

- 就绪态——>运行态：处于就绪态的进程获得处理机进入运行态
- 运行态——>就绪态：处于运行态的进程时间片用完后，让出处理机进入就绪态
- 运行态——>阻塞态：进程请求除处理机外的其他资源，此时运行态进入阻塞态（系统调用请求操作系统提供服务，这是一种特殊的、由运行用户态程序调用操作系统内核过程的形式）
- 阻塞态——>就绪态：进程等待其他资源的获得，如IO资源、或者中断结束

相互转换



进程控制

进程的创建

- 分配进程标识号，申请PCB（PCB是有限的）
- 为进程分配资源，为程序和数据以及用户栈分配必要的内存空间
- 初始化PCB，包括初始化标志信息、初始化处理机状态信息、初始化处理机控制信息、设置进程的优先级
- 若进程就绪队列可以接纳新进程，进程就进入就绪态

进程的终止

结束分类

- 正常结束：进程的任务已经完成并且准备退出运行
- 异常结束：进程正在运行，出现了某些异常事件，导致程序无法继续运行（存储区越界、保护错、非法指令、特权指令错、IO故障等）
- 外界干预：进程应外界请求终止运行

结束过程

- 根据被终止进程的标识符，检索PCB，读取进程状态
- 若进程处于运行态，终止运行，剥夺处理机
- 终止进程之下的子进程
- 该进程拥有的全部资源还给父进程或者操作系统
- 将PCB从队列中删除

进程的阻塞和唤醒

阻塞原语执行过程

- 找到要被阻塞进程标识号对应的PCB
- 若该进程处于运行态，保护其现场，将其状态转换为阻塞态，停止运行
- 将PCB插入相应时间的等待队列

阻塞是一种自主行为，自我阻塞

唤醒原语的执行过程

- 找到等待队列中进程相应的PCB
- 将其从等待队列中移出，置其状态为就绪态
- 将PCB插入就绪队列，等待调度程序调度

唤醒是被相互有联系的其他进程进行唤醒

进程切换

- 进程切换是在内核态下完成的

过程

- 保存处理机上下文，包括程序计数器和其他寄存器
- 更新PCB信息
- 把进程的PCB移入相应的队列，如就绪、在某时间阻塞等队列
- 选择另一个进程执行，更新其PCB
- 更新内存管理的数据结构
- 恢复处理机上下文

2.1进程与线程（中）

进程的组织

- 进程是一个独立的运行单位，也是操作系统进行资源分配和调度基本单位，由以下三部分构成
- 进程控制块
  - 进程描述信息
    - 进程标识符：标志进程
    - 用户标识符：进程归属的用户，主要为共享和保护服务
  - 进程控制和管理信息
    - 进程当前状态：描述进程状态信息
    - 进程优先级：描述进程枪战处理机优先级
    - 代码运行入口地址
    - 程序的外存地址
    - 进入内存时间
    - 处理机占用时间
    - 信号量使用
  - 资源分配清单
    - 用以说明有关内存地址空间或者虚拟地址空间状况，所打开的文件的列表和所使用的的输入/输出设备信息
    - 代码段指针、数据段指针、堆栈段指针、文件描述符、键盘、鼠标
  - 处理机相关信息
    - 处理机中各寄存器的值
    - 通用寄存器值、地址寄存器值、控制寄存器值，标志寄存器值，状态字
- 程序段：能被进程调度程序调度到CPU执行的程序代码段
- 数据段：进程对应的程序加工处理的原始数据或者程序执行时产生的中间或者最终结果
- 进程的组织方式
  - 链接方式
    - 按照进程状态将PCB分为多个队列
    - 操作系统持有指向各个队列的指针
  - 索引方式
    - 根据进程状态的不同,建立几张索引表
    - 操作系统持有指向各个索引表的指针

进程的通信

- 共享存储
  - 通信进程之间存在一块可以被直接访问的共享空间
  - 低级方式：基于数据结构共享
  - 高级方式：基于存储区共享

操作系统只负责为通信进程提供可共享使用的存储空间和同步互斥工具，数据交换则由用户自己安排读/写指令完成

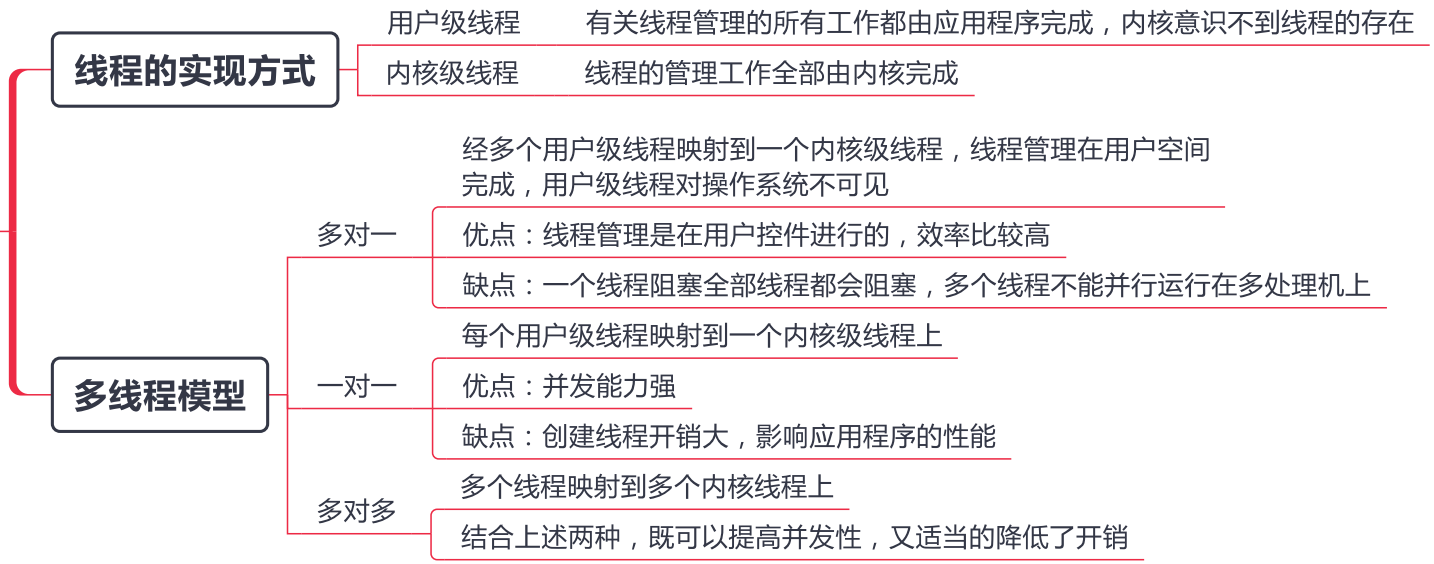
- 消息传递
  - 进程间的数据交换是以格式化的消息为单位的，进程通过系统提供的发送消息和接收消息的两个原语进行数据交换
  - 直接通信方式：发送进程直接发送消息给接收进程，并将它挂在接收进程的消息缓冲队列上，接收进程从消息缓冲队列中取得消息
  - 间接通信方式：发送进程把消息发送给某个中间实体，接收进程从中间实体中获得消息　例如电子邮件系统
- 管道通信
  - 发送进程以字符流形式将大量数据送入写管道，接收进程从管道中接收数据
    - 当管道写满时，写进程的write()系统调用将被阻塞，等待读进程将数据取走
    - 当读进程将数据全部取走后，管道变空，此时读进程的read()系统调用将被阻塞
  - 功能：互斥、同步、确定对方存在
  - 半双工通信，不可以同时读和写
  - 限制管道的大小
  - 管道变空的时候阻塞读进程

管道中的数据被读取后会马上被丢弃

线程概念和多线程模型

- 线程基本概念
  - 减小程序在并发执行时所付出的时空开销，提高操作系统的并发性能
  - 引入线程后，进程只作为系统资源的分配单元，线程作为处理机的分配单元
- 线程与进程的比较
  - 调度
    - 传统中进程是资源和独立调度的基本单位
    - 引入线程后，进程是独立调度的基本单位，线程是资源的基本单位
    - 不同进程的线程切换会引起进程切换
  - 拥有资源　进程是资源分配的基本单位
  - 并发性　引入线程后，进程可以并发执行，多个线程之间也可以并发执行，提高了系统的吞吐量
  - 系统开销　同一进程的线程切换要比进程切换开销小的多
  - 地址空间和其他资源　进程的地址空间之间相互独立，统一进程的各线程之间共享进程的资源，某进程的线程对其他进程不可见
  - 通信方面
    - 进程间通信需要进程同步和互斥手段的辅助，保证数据的一致性
    - 线程间可以直接读/写进程程序段来进行通信
- 线程属性
  - 不拥有系统资源，拥有唯一标识符和线程控制块
  - 不同的线程可以执行相同的程序，同一个服务程序被不同用户调用时，操作系统将其创建为不同线程
  - 统一进程的线程共享该进程拥有的全部资源
  - 线程是处理机的独立调度单位
  - 线程也有生命周期，阻塞，就绪，运行等状态
  - 多CPU计算机中,各个线程可占用不同的CPU
  - 每个线程都有一个线程ID、线程控制块（TCB）
  - 切换同进程内的线程,系统开销很小
  - 切换进程,系统开销较大
  - 由于共享内存地址空间,同一进程中的线程间通信甚至无需系统干预

2.1进程与线程（下）


线程的实现方式
用户级线程
有关线程管理的所有工作都由应用程序完成，内核意识不到线程的存在
内核级线程
线程的管理工作全部由内核完成
多线程模型
多对一
经多个用户级线程映射到一个内核级线程，线程管理在用户空间完成，用户级线程对操作系统不可见
优点：线程管理是在用户控件进行的，效率比较高
缺点：一个线程阻塞全部线程都会阻塞，多个线程不能并行运行在多处理机上
一对一
每个用户级线程映射到一个内核级线程上
优点：并发能力强
缺点：创建线程开销大，影响应用程序的性能
多对多
多个线程映射到多个内核线程上
结合上述两种，既可以提高并发性，又适当的降低了开销

2.2处理机调度（上）

- **调度的概念**
  - 调度的基本概念：合理的对进程进行处理机分配
  - 调度的层次
    - 作业调度（高级调度）：从辅存中选择作业送入内存，每个作业只调入一次，调出一次
    - 中级调度（内存调度）：提高内存利用率和系统吞吐量，将暂时不能运行的进程调至外存，使其进入挂起态。或者将已经具备运行条件的进程调入内存，修改其状态为就绪态
    - 进程调度（低级调度）：按照某种策略或者方法从就绪队列中选取一个进程，将处理机分配给它（最基本的调度，频率很高）
  - 三级调度的联系
    - 作业调度为进程活动做准备，进程调度使进程正常活动起来，中级调度将暂时不能运行的进程挂起，中级调度处于作业调度和进程调度之间
    - 作业调度次数少，中级调度次数略多，进程调度频率最高
    - 进程调度使最基本的，不可或缺的
- **调度的时机、切换与过程**
  - 不能切换的情况
    - 处理中断过程
    - 进程在操作系统内核程序临界区的时候
    - 其他需要完全屏蔽中断的原子操作过程
  - 可以切换的情况
    - 发生引起调度条件且当前进程无法继续进行
    - 中断处理结束或者自陷处理结束
- **进程调度方式**
  - 非剥夺调度方式
    - 如果想将处理机分配给一个更高优先级的进程，必须要等待当前占用处理机的进程释放处理机后才能将处理机分配给更高优先级进程
    - 实现简单 开销小 适合大多数批处理系统
    - 不适用于分时系统，和大多数实时系统
  - 剥夺调度方式
    - 如果有更高级进程请求处理机，暂停正在执行的进程，将处理机分配给更高级进程
    - 提高系统吞吐率和响应效率
- **调度的基本准则**
  - CPU利用率，尽可能保持CPU处于忙碌状态
  - 系统吞吐量：单位时间内CPU完成作业的数量，调度算法和方式会对吞吐量造成较大影响
  - 周转时间：作业提交到作业完成的时间
    - 周转时间= 作业完成时间 - 作业提交时间
    - 平均周转时间 = 总周转时间/N个作业
    - 带权周转时间 = 作业周转时间/作业实际运行时间
    - 平均带权周转时间 = 总带权周转时间/N个作业
  - 等待时间：作业等待处理机的时间，衡量一个算法优劣，只需要简单的考察等待时间
  - 响应时间：从用户提交请求到系统首次产生响应所用的时间
- **进程的挂起态与七状态模型**
  - 暂时调到外存等待的进程状态为挂起状态（挂起态）
    - 就绪挂起
    - 阻塞挂起
  - “挂起”和“阻塞”的区别　　两种状态都是暂时不能获得CPU的服务，但挂起态是将进程映像调到外存去了，而阻塞态下进程映像还在内存中

2.2处理机调度（下）

典型调度算法

先来先服务算法

- 作业调度 进程调度
- 先来的先分配处理机
- 优点：算法简单 对长作业有利 有利于CPU繁忙型作业（计算型）
- 缺点：效率低 不利于短作业 不利于IO繁忙型作业
- 不会导致饥饿
- 非抢占式的算法

短作业优先算法

- 进程调度
- 优先选择预计运行时间最短的进程
- 优点：平均等待时间 平均周转时间最短
- 缺点：对长作业不利，造成饥饿现象，没有考虑作业的紧迫性，用户可能可以缩短作业预估时间，使得无法做到短作业优先
- 产生“饥饿”现象。如果一直得不到服务，则称为“饿死”
- SJF和SPF（短进程优先（SPF）算法）是非抢占式的算法。但是也有抢占式的版本——最短剩余时间优先算法

优先级调度算法

- 作业调度 进程调度
- 分类
  - 剥夺型 立即停止当前运行进程，将处理机分配给更高优先级进程
  - 非剥夺型 等待当前进程运行完成，然后将处理机分配给更高优先级进程
- 优先级分类
  - 静态优先级：进程创建后无法对优先级进行修改
  - 动态优先级：可以根据进程运行状态，对进程优先级进行动态调整
- 优先级设置原则
  - 系统进程>用户进程
  - 交互型进程>非交互型进程
  - I/O进程>计算型进程（cpu繁忙型）
- 产生“饥饿”现象
- 有抢占式的，也有非 抢占式的

高响应比调度算法

- 响应比=（等待时间+要求服务时间）/要求服务时间 = 1+等待时间/要求服务时间
- 等待时间相同情况下，要求服务时间越短响应比越大，有利于短作业进程
- 要求服务时间相同，作业响应比由其等待时间决定，等待时间越长响应比越高，实现先来先服务
- 对于长作业，作业的响应比可以随等待时间的增加而提高，等待时间足够长时，其响应比可以升到很高，从而获得处理机
- 不会导致饥饿
- 非抢占式的算法

时间片轮转算法

- 使用与分时系统，使用时间片，就绪进程按照到达先后排成队列，依次在时间片内占用处理机，时间片到达时就释放处理机
- 时间片选择很重要，过大就变成了先来先服务，过短又变成了短作业优先
- 时间片影响因素：系统响应时间，就绪队列中的进程数目和系统的处理能力
- 不会导致饥饿
- 抢占式

多级反馈队列调度算法（融合前面几种算法）

- 实现思想
  - 设置多个就绪队列，为每个队列设置不同的优先级，优先级依次递减
  - 每个队列中的时间片各不相同，时间片依次递增
  - 每个队列按照先来先服务原则进行进程排队，若规定时间片内没有完成，就将进程放入下一级队列
  - 只有到高级队列为空的时候，低等级队列才能开始调度。
- 优点
  - 终端型作业用户：短作业优先
  - 短批处理作业用户：周转时间较短
  - 长批处理作业用户：讲过前面几个队列得到部分执行，不会长期得不到处理
- 产生“饥饿”现象
- 抢占式

2.3进程同步

基本概念

- 临界资源
  - 一次只允许一个进程使用的资源（打印机，特殊变量，数据）
  - 临界资源的访问过程
    - 进入区：检查进程是否可以进入临界区
    - 临界区：可以访问临界资源的代码
    - 退出区：将正在访问临界区的标志清除
    - 剩余区：代码中的其余部分
- 同步：直接制约关系，为了完成某种过任务而建立的多个进程，相互合作，所以要相互进行通信同步
  - 遵循的原则
    - 空闲让进：临界区空闲时，可以允许一个请求进入临界区的进程立即进入临界区
    - 忙则等待：已有进程进入临界区后，其他试图进入临界区的进程必须等待
    - 有限等待：对于请求访问临界区的进程，在有限时间内进入临界区
    - 让权等待：进程不能进入临界区的时候，应当立即释放处理机
- 互斥：间接制约关系，当一个进程访问临界资源的时候，其他进程不能访问

实现临界区互斥的基本方法

- 软件实现方法
  - 单标志法
    - 两个程序进程交替进入临界区
    - 优点：实现简单
    - 缺点：可能会违背空闲让进，造成资源无法充分利用
  - 双标志法先检查
    - 每个进程访问临界区资源前，先检查临界资源是否被访问，如果空闲才能进入
    - 优点：不用交替进入可以连续使用
    - 缺点：两个进程可能同时进入临界区，违背忙则等待
  - 双标志法后检查
    - 先设置自己标志，表明自己想要进入，检查对方标志，如果对方也要进入，那么就等待否则就进入
    - 优点：不会导致两个进程都进入临界区
    - 缺点：双方可能会互相谦让，导致饥饿现象
  - 皮特森算法
    - 防止两个进程无限期等待，在算法三的基础上增加一个标志位，从而防止饥饿
    - 优点：解决了饥饿现象
    - 缺点：算法复杂
- 硬件实现方法
  - 中断屏蔽法
    - 对中断进行屏蔽、关中断
    - 优点：关中断非常方便
    - 缺点：限制了处理机交替执行程序的能力
  - 硬件指令法
    - 读出指定标志后，将该标志置为真

优缺点

- 优点
  - 适用于任意数目的进程
  - 简单且容易验证正确性
  - 支持进程内有多个临界区
- 缺点
  - 不能实现让权等待
  - 可能会导致饥饿现象

信号量

- 整形信号量
  - wait: 资源-1 signal=资源+1
  - 没有遵循让权等待机制，会导致进程处于“忙等”状态
- 记录型信号量
  - 记录型信号量不存在“忙等”现象，除了需要一个用于代表资源树木的整型变量value外，在增加一个进程链表L，用于链接所有等待该资源的进程
- 利用信号量实现同步
  - 设S为进程P1和P2同步的公共信号量，初值为0，通过设置S的值可以使得P1与P2按照一定顺序执行
- 利用信号量实现互斥
  - 通过设置S的值，可以实现进程对临界资源的互斥访问
- 利用信号量实现前驱关系
  - 通过设置不同的进程运行结束后，产生不同的信号量，从而可以使得目标进程运行，从而实现前驱关系

管程

- 定义
  - 一组数据以及定义在这组数据之上的对这组数据的操作组成的软件模块，这组操作能初始化并改变管程中的数据和同步进程
- 组成
  - 局部于管程的共享结构数据说明
  - 对该数据结构进行操作的一组过程
  - 对局部于管程的共享数据设置初始值的语句
- 基本特性
  - 局部于管程的数据只能被局部于管程内的过程所访问
  - 一个进程只有通过调用管程内的过程才能进入管程访问共享数据
  - 每次仅允许一个进程在管程内执行某个内部过程

2.4 死锁

- **死锁的概念**
  - 死锁的定义：多个进程因为竞争资源造成的一种僵局，没有外力作用，这些进程都无法向前继续推进
  - 死锁产生的原因
    - 系统资源的竞争
    - 进程推进顺序非法
    - 死锁产生的必要条件
      - 互斥条件：进程对分配的资源进行排他性控制
      - 不可剥夺条件：进程获得资源在未使用完之前，不能被其他进程强行夺走
      - 请求并保持条件：进程已经保持了至少一个资源，提出新的资源请求，而该资源已经被其他进程占有，此时该进程被阻塞，但是自己已经获得的资源保持不放
      - 循环等待条件：你等我释放 我等你释放
- **死锁的处理策略**
  - 死锁预防
    - 破坏四个必要条件中的一个或几个，防止死锁
    - 资源分配保守，宁可资源闲置
    - 一次性请求所有资源，资源剥夺，资源按序分配
    - 优点：适用于突发式处理的进程，不必进行剥夺
    - 缺点：效率低，进程初始化时间长，剥夺次数过多，不变灵活申请新资源
  - 避免死锁
    - 在资源的动态分配中，用某种方法防止系统进入不安全状态，避免死锁
    - 运行过程中预测分配资源是否会死锁
    - 寻找可能的安全序列
    - 优点：不必进行剥夺
    - 缺点：必须知道将来的资源需求，进程不能被长时间阻塞
  - 死锁的检测及解除
    - 允许进程死锁，通过检测及时的判断死锁，然后对其进行解除
    - 宽松，只要允许就分配资源
    - 定期检查是否死锁
    - 优点：不延长初始化时间，允许对死锁进行现场处理
    - 缺点：通过剥夺解除死锁，造成损失
- **死锁预防**
  - 破坏互斥条件：某些资源只能被互斥访问，并且某些情况下必须保护互斥性
  - 破坏不剥夺条件
    - 释放已经占有的资源
    - 特点：增加系统开销 实现复杂 降低吞吐量
    - 用于状态易于保存和恢复的数据（CPU的寄存器及内存资源）
  - 破坏请求并保持条件
    - 一次性申请完所需要的全部资源
    - 特点：实现简单，但是资源被严重浪费，甚至可能导致进程饥饿
  - 破坏循环等待条件
    - 采用顺序资源法，对进程进行顺序推荐
    - 特点：进程编号必须稳定，可能会导致资源浪费，并且不利于用户编程
- **死锁避免**
  - 系统安全状态：按照某种方式分配资源后，是否会导致死锁，如果会导致死锁，那么就是不安全状态，反之就是安全状态
  - 银行家算法：思想：通过计算当前资源的不同分配方式，从而预测系统是否会进入不安全状态　就像是银行贷款，是否会导致银行没有足够的资金对外出借
- **死锁的检测和解除**
  - 资源分配图：圆圈表示进程，框表示一类资源，进程到资源的有向边称为请求边，资源到进程的边称为分配边
  - 死锁定理
    - 在资源分配图中找到分配满足的进程，然后消去其请求边与分配边
    - 如果最后所有边都可以被消去，那么就是可以简化的，不存在死锁，反之存在死锁
  - 死锁解除
    - 资源剥夺法：挂起某些死锁进程，抢占资源，将这些资源分配给其他死锁进程，但是要防止挂起时间过长
    - 撤销进程法：强制撤销部分甚至全部死锁进程，并且剥夺他们的资源，撤销原则可以根据优先级和撤销进程的代价进行
    - 进程回退法：让一个或者多个进程回退到足以回避死锁的地步，进程回退时自愿释放资源而非被剥夺。要求系统保持进程历史信息，设置还原点
- **死锁、饥饿、死循环的区别**
  - 死锁：各进程互相等待对方手里的资源，导致各进程都阻塞，无法向前推进的现象
  - 饥饿：由于长期得不到想要的资源，某进程无法向前推进的现象
  - 死循环：某进程执行过程中一直跳不出某个循环的现象